22 385-201 768606 A

PATENT APPLICATION

ATTORNEY DOCKET NO. A1041C1 US

METHOD AND APPARATUS FOR EMBEDDING CHAT FUNCTIONS IN A WEB PAGE

Andrew T. Busey
Gerald H. Weghorst, Jr.

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of copending United States Patent Application serial number 08/741,470, filed October 30, 1996.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to network communications, and more particularly to synchronized browse and chat functions on a computer network.

Description of Related Art

The location and exchange of data over computer networks is controlled by various network protocol. For example, the World Wide Web (hereinafter "Web") is a system of communications protocols that presents information in documents that are capable of being

linked to other documents. The documents are stored in a distributed manner across the Internet on the networked computers, and are accessed using programs known as browsers.

The Web is a system of protocols exchanged between a host computer running an application, known as a server, that delivers Web documents, and a user's computer, known as the client. Web documents are created using a markup language known as HTML, or Hypertext Markup Language. Hyperlinks are used to connect a document on one host computer to a document on another host computer. The following HTML paragraph is illustrative.

```
<P>
Welcome to the home page of <B> ichat, Inc.</B>. We develop <A
HREF="../products/index.html">software</A> that expands the
functionality and accessibility of real-time Internet chat systems.
```

The HTML tag "<A HREF=" instructs the browser to create a link to a web page referenced by the embedded Uniform Resource Locator ("URL"), which is a type of address, and to use the word "software" embedded between the tags "> ... </A>" as the hyperlinked word. The link may be a target, which is a word or phrase in another section of the same Web page; a relative link, which is another Web page within the current site, either forward or backward relative to the current page; or an external or absolute link, which is a Web page on another host.

The dominant transfer protocol in use on the Web is HTTP, which stands for Hypertext Transfer Protocol. HTTP sits on top of TCP/IP and is a stateless protocol designed to transfer documents at a high rate of speed. As a stateless system, HTTP does not retain any information from one document transfer to the next. If additional documents are needed, each additional document must be transferred by opening a new HTTP connection, requesting the document, delivering the document, and closing the connection.

Although generally successful for many Web functions, the HTTP protocol is ineffective for enabling real-time functions such as chat over networks such as the Internet. However, chat is available over the Internet using the Internet Relay Chat protocol, or IRC. IRC uses the TCP/IP protocol in a client/server model. One IRC client initiates a channel by connecting to an IRC enabled server, which may or may not be on the same host as the Web server. Other clients join the channel by typing a join IRC command. The IRC server mediates the channel, passing each message to all channel members or to particular channel members, as determined by the originator of the message.

As initially implemented, IRC was of limited usefulness to users who wished to coordinate their chats with Web browsing. However, a technique known as integrated HTML chat has emerged for facilitating coordination of chats with browsing. In integrated HTML chat, chat is incorporated into the HTML frame and has the appearance of being embedded. The user receives an HTML page that contains the chat window, types his or her reply to a message in the chat window, and sends the revised page back to the originating server. Unfortunately, integrated HTML chat is a limited and inflexible technique. The chat and browser applications run independently of one another, relying on user interaction at particular points in time to achieve browse-chat coordination. Unfortunately, independence of operation causes the browser and chat applications to be generally uncoordinated, and the need for the user to coordinate their operation is inconvenient.

SUMMARY OF THE INVENTION

The present invention advantageously embeds chat functions into Web pages using the well-defined application program interfaces of various web browsers. More specifically, one embodiment of the invention is a method for embedding chat functions in a Web page. A chat client is linked to a browser through the application program interface of the browser. The browser is resident on a client computer having a client display device, and a server furnishes commands to the browser to establish browser and chat regions on the client display

device. The same or a different server furnishes browser content to the browser, which displays the browser content in the browser region of the client display device. A chat server furnishes chat content to the browser which invokes the chat client through the application program interface. The evoked chat client displays the chat content in the chat region of the client display device through the browser.

Another embodiment of the invention is a system for embedding chat functions in a Web page. A client computer includes a client display device, an installed browser application having an application program interface, and an installed chat application. The chat application is linked to the browser application through the browser's application program interface. A server is programmed to furnish commands to the browser application for establishing browser and chat regions on the client display device. The same or a different server is programmed to furnish browser content to the browser application, the client computer being programmed by the browser application to display the browser content in the browser region of the client display device. A chat server is programmed to furnish chat content to the browser application, the client computer being programmed by the browser application to invoke the chat client through the application program interface, and the client computer being programmed by the chat client to display the chat content in the chat region of the client display device.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings, in which like reference characters indicate like parts:

Figure 1 is a schematic diagram of network protocol connections between clients and a host in accordance with the present invention;

Figure 2 is a flow chart of a method for real time network chat in accordance with the present invention;

Figure 3 is a schematic diagram of how a hyperlink functions during a real time network chat, in accordance with the present invention;

Figures 4A, 4B, 4C, 4D, 4E, 4F, 4G, 4H, 4I and 4J are pictorial representations of user displays, in accordance with the present invention;

Figure 5 is a schematic diagram of a world topography;

Figure 6 is a block schematic diagram showing a state of a two room world and the relationships between a chat server, a Web server, and elements of the user's display;

Figure 7 is a block schematic diagram showing another state of a two room world and the relationships between a chat server, a Web server, and elements of the user's display;

Figure 8 is a schematic diagram of an implementation of embedded chat for the Netscape Navigator browser, in accordance with the present invention; and

Figure 9 is a block schematic diagram showing a server architecture suitable for handling a chat session using plug-ins, ActiveX controls, and Java applets.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 and Figure 2 show a process for real-time conferencing across the Internet. The process begins with a user who launches a chat session from his or her computer, preferably from a browser application running on the computer, by running an application called a real time markup ("RTM") chat client. The computer operating system ("OS") causes a two-way TCP/IP connection to be established between the client computer and a host computer for the chat session, while the RTM chat client causes a real time protocol ("RTP") connection, typically full duplex, to be established between the RTM chat client and

a real time server on the host. Other users join the chat session by establishing TCP/IP connections and launching their own RTM chat clients. Figure 1 shows three RTM chat clients 110, 120 and 130, which run on top of respective TCP/IP clients 112, 122 and 132. The TCP/IP connections are established with a host computer, which runs TCP/IP host software 140 and typically hosts several different types of servers. Figure 1 illustratively shows four servers, an HTTP server 142, a telnet/chat server 144, an FTP server 146, and an Internet Relay Chat ("IRC") server 148. Typically, a variety of other server types reside on the host computer as well, including, for example, Gopher, Usenet and WAIS.

A real time chat client is any client capable of sustaining what appears to a user to be real time chat. The effect of real time is created by using as the RTP a continuously open connection protocol such as, for example, a continuously open streaming protocol such as telnet or a continuously open connection packet protocol such as IRC. Telnet is a well known streaming protocol used to establish bi-directional continuously opened sockets and full duplex data transmission to achieve real time communications. The telnet protocol is an industry standard. UNIX hosts are generally provided with telnet servers as part of their operating systems. Other examples of continuously opened connection streaming protocols include UDP, or Universal Data Protocol, and a variety of proprietary protocols. IRC is a well known packet protocol used to establish bi-directional continuously opened sockets and full duplex data transmission to achieve real time communications. The IRC protocol is an industry standard, fully defined in RFC 1459. In contrast, the HTTP protocol defines a transactional half-duplex data transmission. HTTP connections are opened and closed as documents are requested and sent. Real time communication is not realized.

A markup language is any language that enables document formats to be defined, and may also enable hyperlinks to be embedded in documents. The most popular markup language in use on the Web is HTML, which supports embedded hyperlinks, various font styles such as bold and italics, and various MIME (Multipurpose Internet Mail Extension) file types for text and embedded graphics, video and audio.

Figure 2 shows what happens when a RTM chat client is launched. Illustratively, the chat client in Figure 2 is a telnet HTML chat client and the host includes a telnet server and a server-side application known as a chat server that enables communication between two or more chat clients. While Figure 2 shows use of the telnet protocol and a compatible chat server, the IRC protocol and an IRC chat server may be used if desired, as well as any other continuously open bi-directional connection chat client - server types and compatible chat server applications. Chat servers are well known; for example, the telnet protocol and proprietary chat server software is commonly used by commercial BBS services, and the IRC protocol and IRC server side chat applications are common in many UNIX environments. While Figure 2 also shows use of HTML, other markup languages may be used if desired.

After the TCP/IP and telnet connections are made (step 200), the telnet HTML chat client immediately begins to receive any messages being posted by the chat server, and may send messages to other telnet HTML chat clients through the chat server or remain idle in the event that no messages are being sent or received. While non-HTML telnet clients may also be connected to the chat server, they will not be capable of displaying the incoming data with fidelity because they will not be able to properly parse it.

Messages outgoing from the telnet chat client are processed as follows. The telnet chat client is designed either to send each keystroke to the host either individually or in groups. In either case, the telnet chat client appends the keystroke(s) to a TCP/IP header and the resulting packet is sent to the chat host (step 220). The chat host parses the incoming data in real time (step 222). If the chat host detects a telnet escape sequence (step 224), it processes the detected escape sequence (step 226). If the chat host is set to a mode for processing HTML tags (step 227) and detects a server-executable HTML tag (step 228), it processes the detected HTML tab as appropriate (step 229). If the chat host does not detect a telnet escape sequence and either is not in an HTML tag detect mode or does not detect a server HTML tag if in an HTML tag detect mode, the chat host simply posts the data (step 230) to all connected telnet clients or to a specific or ones of connected telnet clients if so

instructed by the chat server. Connected telnet clients that are not HTML enabled simply display any HTML tags as they are received. However, connected telnet HTML clients recognize and respond to the HTML tags in the data.

Messages outgoing from an IRC chat client are processed in a slightly different manner. An IRC packet is the entire series of keystrokes preceding a carriage return. An IRC chat client appends the IRC packet or in some cases breaks up the IRC packet into sub-packets and appends each sub-packet to a TCP/IP header, and the resulting TCP/IP packet is sent to the IRC chat host. The IRC chat host parses the incoming data in real time, processing any IRC headers and handling the appended data accordingly.

The telnet chat client processes incoming messages containing HTML tags as follows. The telnet chat client parses the incoming data (step 210) to distinguish between HTML tags and characters to be displayed. If a client-executable HTML tag is detected (step 212), the tag is processed as appropriate (step 214). If a client HTML tag is not detected (step 212), the incoming data is displayed on the chat screen of the telnet chat client computer (step 216). In either case, the telnet chat client then looks for more data to process (step 218), and either resumes parsing or idles if no incoming or outgoing message is present.

The telnet connection is terminated either by the client or the host. Termination is done by releasing the socket for the connection, in a manner well known in the art.

An example of a real time chat session among chat clients using HTML is as follows.

<Sarah> Hi everyone! I found a *great* web site. **Check out** the ichat site.
<Sam> Thanks for the info, Sara. I'm going to check out the site now. Bye.

This text appears on the screens of the HTML chat clients who are members of the chat session.

When Sarah types her message, she uses either macros or HTML itself to cause the word "great" to appear in an italics font style, the phrase "Check out" to appear in a bold font style, and to create the hyperlink ichat site. Sarah's chat client software sends the following illustrative data stream to members of the chat session via the host.

Hi everyone! I found a <I> great</I> web site. <B> Check out</B> the <A HREF="http://www.ichat.com"> ichat site.</A>

The HTML chat client software displays Sara's message as it is typed in a normal font, until the "<I>" tag is detected (the sharp brackets connote an HTML control). The characters "great" are displayed as they are typed in an italics style font until the "</I>" tag is detected, after which subsequent characters are again displayed at they are typed in a normal font. When the "<B>" tag is detected, the subsequent characters "great" are displayed as they are typed in a bold font until the "</B>" tag is detected, after which subsequent characters are again displayed at they are typed in a normal font. When the tag "<A HREF="http://www.ichat.com">" is detected, Sam's software responds by linking the URL "http://www.ichat.com" to the text following the tag, until the tag "</A>" is detected. Hence, the URL "http://www.ichat.com" is linked to the hyperlink ichat site. This hyperlink is displayed as its characters are typed in a underlined and colored font until the "</A>" tag is detected, after which any subsequent characters are displayed at they are typed in a normal font.

Sam responds to Sara's message with his message, and then simply clicks on the hyperlink "ichat site" in his chat window using either his mouse or keyboard navigation. This action launches Sam's Web browser, if it is not already running. Sam's Web browser takes him to the ichat home page, without need for Sam to enter a URL.

The manner in which hyperlinks function in a chat session among RTM chat clients is shown in more detail in Figure 3. The two way arrow between RTM chat client 314 in client 310 and a real time server 324 in host 320 represents a bi-directional TCP/IP - real time

protocol communications channel. The two way arrow between RTM chat client 334 in client 330 and the real time server 324 in host 320 also represents another bi-directional TCP/IP - real time protocol communications channel. The one way arrows between web browser 332 in the client 330 and HTTP server 342 in host 340 represent respective one way TCP/IP HTTP (transactional) protocol communications channels. The host 310 need not include a Web browser, the host 320 need not include an HTTP server 322, and the host 340 need not include a real time server 340.

RTM chat client 314 (*e.g.* Sarah) creates a message that includes an embedded hyperlink, and sends that message through the real time server 324 (action "A") to the RTM chat client 334 (*e.g.* Sam) (action "B"). Under some circumstances the real time server 324 acts on the embedded hyperlink, although in the example of Figure 3 the real time server 324 merely posts the message to the RTM chat client 334. Note that other actions that may be occurring, such as echo of the message back to the RTM chat client 314 and communication of the message to other joined chat clients, are omitted for clarity. The client 330 (*e.g* Sam) then causes his Web browser 332 to access the URL associated with the hyperlink embedded in the chat message (*e.g.* ichat site) (action "C"). Action "C" is performed in any suitable manner. For example, if the Web browser 332 is inactive, the RTM chat client 334 simply launches the Web browser 332 using the URL associated with the hyperlink as a command line argument. If the Web browser 332 happens to be running, the RTM chat client 334 communicates the page request to the Web browser 332 using any suitable interface protocol such as the DDE protocol, which is standard in such operating systems as the Microsoft® Windows® Version 3.1 operating system and the Microsoft® Windows® 95 operating system. Newer protocols and methods suitable for having the RTM chat client 334 cause the Web browser 332 to acquire a Web page include plug-in technologies, ActiveX technologies, and Java technologies. The Web browser 332 makes a TCP/IP connection with the HTTP server 342 (or any other HTTP server, including HTTP server 322) and Web browser 332 makes a request for a Web page (action "D") by sending the URL associated with the embedded hyperlink. The HTTP server 342 responds by delivering the requested Web page (action

"E"), and the TCP/IP connection between the Web Browser 332 and the HTTP server 342 is terminated. Meanwhile, the bi-directional TCP/IP - real time protocol communications channels between the RTM chat client 314 and the real time server 324, and between the RTM chat client 334 and the real time server 324 remain open if desired to continue the chat session.

Figure 6 shows a real time chat server 610, which not only maintains the chat session as does the real time server 324 of Figure 3, but also synchronizes the browse and chat functions by dynamically linking the browser and chat applications to allow the contents of the browser window and the chat window to change in a coordinated manner. In this way, multiple users' browsers may be connected into one powerful distributed chat/HTTP server and all such users are able to fully interact with one another in a coordinated manner via type-written messages, HTML web documents, and file transfers.

A useful metaphor for synchronized chat and browse functions is that of a visitor able to move from room to room of a building, each room containing remarkable things and a group of people engaged in a spirited discussion about the things in their room. The visitor may remain in the present room and continue the present discussion while looking at the remarkable things on display in the present room, or may peak into a different room and continue the present discussion while looking at the remarkable things on display in the different room, or may move to a different room and join the discussion in progress in that different room, or may follow another visitor to a different room and join with the other visitor in the discussion in progress in that different room. In practice, the real time chat server 610 synchronizes the browse and chat functions when the user, or visitor of the metaphor, moves into another room, so that not only does the browser content change but the chat also changes in a coordinated manner.

Consider the following example of communications on the Web, which shows the usefulness of a dynamic link between the browse and chat functions. A user loads a first chat-

enabled Web page and wishes to join a chat associated with the first Web page. The user merely clicks on a chat button to display a chat associated with the first Web page. Preferably the chat window is embedded in the Web browser window, although the two windows may be separate if desired or the browser window may be embedded in the chat window.

If the user learns of a second Web page of interest from another chat member during the chat, the user need only click on the hyperlink provided in the chat window to examine the second Web page while continuing to participate in the chat associated with the first Web page. If the second Web page is chat-enabled and the user wishes to join a chat associated with the second Web page, the user need only click on the chat button in the second Web page. Lets say that this new site has a "speaker" who presents a variety of "slides," which are different Web pages on the speaker's site or on other sites or are sections of the current Web page, and discusses each slide with users in the chat group. A user might ask a question about one of the slides, to which the speaker is able to respond by simply answering the user's question in the chat window, or by showing other slides to the user individually or with the audience and discussing the other slides in the chat window, or by taking the user individually or with the audience to a different Web page on or off the current site either to continue the chat by discussing the new Web page or to join the chat associated with the new Web page, or by suggesting that the user go to a different Web page to join a chat associated with the new Web site.

A sequence of events illustrating some of these capabilities is shown in Figure 4. Figure 4A shows the screen of a user Ursula who wishes to join a chat in progress on the subject of where to go to fix computers. The user launches her Web browser and brings up an HTML page 410 in her browser window having a schedule of computer chats. Illustratively, one of the scheduled chats is on sharing ideas about where to go to fix computers is in progress. To join in that chat, the user clicks the hyperlink "CHAT" (411).

Figure 4B shows the result of the user's having clicked on "CHAT" (411). A chat window opens containing the selected chat session 412. The chat window containing the chat session 412 is embedded in the browser window displaying the page 410. The user announces to the chat group by typing (not shown) into a message entry area 414 that she needs advice on fixing her computer, and one of the chat participants, Able, responds in the chat session 412 by recommending the Fixit! repair advisor service. Able also creates the hyperlink "Fixit! for the user and the chat group generally. Ursula thanks Able and clicks on the hyperlink "Fixit!" (action 416) in the chat session 412.

Figure 4C shows the result of the user's having clicked on "Fixit!" (action 416). A Fixit! welcome page 420 displays which contains a list of computer brands, and the user is able either to enter into a chat about any of the computer brands by clicking on the hyperlink for that computer brand on the page 420, or to request help from the receptionist Rob by clicking on the hpyerlink "Chat with Rob the Receptionist" on the page 420.

Figure 4D shows the result of the user's having clicked on "Chat with Rob the Receptionist" (action 412). A chat window appears showing a synchronized chat session 422. The chat window containing chat session 422 is embedded in the browser window containing the HTML page 420. The chat window containing chat session 422 also contains a message entry area 424. Since Ursula's computer type is not listed, she joins the chat session in progress by using the message entry area 424 (Figure 4C) to ask the receptionist if Fixit! can help repair her computer. The receptionist Rob replies in the chat session 422 that the user's ABC brand computer is the same as a BCD brand computer, and offers to move her to a BCD computer repair advisor for help.

Figure 4E shows a Fixit! repair advisor welcome page 440 appearing in the browser window along with a synchronized chat session 442 in the embedded chat window, which result from Rob's having moved Ursula to BCD Computer repair advisor Ralph. Illustratively, pages 420 and 440 are on the same site, *viz.* the Fixit! site. The chat window

containing the chat session 442 also contains a message entry area 444. An advertising banner 446 appears in a banner window above and adjacent to the browser window containing page 440. The general type of banner is synchronized with the chat, here BCD Computer Inc. advertisements where the chat is about fixing BCD brand computers, and the specific message of the banner changes either periodically or in some other suitable manner, as by context sensitivity. The repair advisor Ralph greets Ursula, Ursula describes the problem, and Ralph suggests a look at a particular HTML page to help identify exactly the failed component in Ursula's computer.

Figure 4F shows in the browser window an image 458 of one type of disk drive that might have been installed in Ursula's ABC Computer. The image 458 is part of HTML page 450. Page 450 is not from the Fixit! site. Rather, page 450 is from the site http://www.diskco.com, although it could have been a page from a different server on the Fixit! site, a page from the same server as page 440, or even a different section of the page 440. Page 450 appears with and is synchronized with chat session 442 in the embedded chat window. A new synchronized banner 456 appears in the banner window. As it so happens, the component shown in the image 458 is not the type in Ursula's computer. Ralph suggests a look at another HTML page to help identify exactly the failed component in Ursula's computer.

Figure 4G shows in the browser window an image 468 of another type of disk drive that might have been installed in Ursula's ABC Computer. The image 468 is part of HTML page 460, which also is from the server http://www.diskco.com, although it could have been a page from a different site or server. Page 460 appears with and is synchronized with chat session 442 in the embedded chat window. A new synchronized banner 466 appears in the banner window. As it so happens, the component shown in the image 468 is not the type in Ursula's computer. Ralph suggests a look at a Fixit! promotional page while he decides on a course of action.

Figure 4H shows in the browser window an image 478, which is a promotional graphic. The promotional graphic 478 is part of HTML page 470, which is Ralph's page on the Fixit! site. The new page 470 and the old chat window 442 are synchronized. Another new synchronized banner 476 appears in the banner window. Unfortunately, none of the components shown in graphics 458 and 468 is in Ursula's computer, and Ralph announces his conclusion in chat session 442 (Figure 4H) that Ursula has an undocumented component in her computer. Ralph decides that the best course of action is for Ursula to follow him to chat with a BCD Computer technician to discuss the problem, and makes a suggestion to this effect.

Figure 4I shows a BCD Computer technical services desk welcome page 480 in the browser window. HTML page 480, which illustratively is on the BCD Computer Web site and not on the Fixit! Web site, is selected by Ralph so that he can discuss Ursula's problem with a BCD Computer technical expert. Page 480 appears with and is synchronized with chat session 442 in the embedded chat window, which contains the message entry area 444. A new synchronized banner 486 appears in the banner window. Since Ralph wants to chat with a BCD Computer technician, he clicks on "Click here" (action 481) on the page 480.

Figure 4J shows the BCD Computer technical services desk welcome page 480 in the browser window. However, since Ralph clicked on the chat hyperlink on page 480 and since Ursula is following Ralph, the new page 480 becomes synchronized with a new chat session 482 in the chat window, which contains the message entry area 484. Illustratively, no banner is associated with either the new page 480 or the new chat session 482, so no banner window appears. Ursula is likely to have her problem resolved now that experts from Fixit! and BCD Computer are both involved in a chat with her about her problem.

Table 1 below summarizes the capabilities showcased in the foregoing example, which constitute a useful set of capabilities for chat-enabled Web servers and clients.

TABLE 1

OBJECTIVE	ACTION	APPEARANCE OF BROWSER AND CHAT WINDOWS
View a new page (different site, same site, or same server)	Select a bookmark, or enter the URL of the new page, or click on a hyperlink in the browser or chat windows.	Desired page displays in the browser window while the current chat session continues in the chat window. If the new page is chat enabled, a hyperlink is shown which enables a new chat session synchronized with the new page to be displayed.
Join a new chat session	Navigate to an adjoining "room" using a direction command, or by clicking on a hyperlink in the chat window, or by using a teleportation command, or by using a goto command to join a specified target.	New chat session displays in the chat window, and a synchronized page displays in the browser window.
	View a new chat-enabled page, then click on the chat hyperlink.	Desired page displays in the browser window, then after the click a new chat session synchronized with the new page displays in the chat window.
Follow another user	Invoker enters the "follow" command followed by the name of the user (the target) the invoker wishes to follow.	The invoker's browser and chat windows are the same as that of the target until either the invoker or the target enters a "stop following" command.
Post an HTML page in another user's browser window	Invoker enters the "suggest" command followed by the name of the target and the URL of the HTML page.	The target's chat window remains the same as the invoker's chat window but the target's browser window contains the page specified by the invoker.

OBJECTIVE	ACTION	APPEARANCE OF BROWSER AND CHAT WINDOWS
Move a target to another chat session and post its associated HTML page to the target's browser window.	Invoker enters the "move" command followed by the name of the target and the path of the new chat room.	New chat session displays in the chat window, and a synchronized page displays in the browser window.

Various implementations of a chat server to synchronize the chat and browse functions are possible and include a variety of command sets as well as linear and object oriented programming techniques. Preferably, the chat server is implemented using object oriented techniques. Users are realized as instantiations of a user object. The user object "resides" in an instantiation of a room (like one of the rooms of the metaphor), which is characterized by a particular browser connection (like the remarkable things of the metaphor) and a particular chat connection (like the discussion group of the metaphor). The user object is fully aware of these browser and chat connections of the room in which it is resident. Like the visitor of the metaphor, the user object is able to move or be moved to another instantiation of a room, which is also characterized by a particular browser connection and a particular chat connection, and becomes fully aware of the new browser and chat connections that characterize the new room. Physically, the user's browser and chat connections change immediately as the user object become aware of the browser and chat connections of the new room.

The room is an element of a world, which is a collection of rooms. The rooms within a particular world define areas of special interest and preferably are arranged in a manner where a user can move easily from one room to another. For example, if the designer of a world is working on a Web site that will showcase five products of a common manufacturer, the world is in one implementation a six room world, one room being a main entry area and

the other five being product rooms. In this implementation, each room contains an HTML page illustrating features of a particular product and a chat area where customers and salesmen discuss the product. When designing a world, take into account how many rooms are needed for the world, what URLs (HTML pages) will be associated with each room, and how the rooms are connected (user traffic flow).

User traffic flow is taken into account when designing a world. User traffic flow is defined by room exits that connect one room to another. For example, the product promotional site mentioned above in one implementation has the six room topology shown in Figure 5, with an entry room 500 having five exits to respectively the five product rooms 510, 520, 530, 540 and 550, and with each of the product rooms having one exit back to the entry room.

Illustratively, a language called LPC is suitable for implementing the chat server, although other languages are suitable as well. LPC is an object-oriented interpreted language that is widely used in multi-user network applications, typically Multi-User Dungeons (MUDs). The chat server is built from a number of core software objects within the LPC framework. These core objects are user objects, connection objects, and room objects.

User objects are used by the chat server to identify the users of the server and to distinguish the individual user preferences on that server. Each of the users of the chat server identifies himself or herself, if desired, by means of name, gender, address, e-mail address, URL, an avatar, and a description. These attributes are stored in the user object and can be queried and viewed by other users of the chat server. The user object also has associated with it the method by which the user connected to the server. The connection method is obtained through the connection objects that the user objects can inherit.

Connection objects give the chat server the capability to provide network connections to a variety of standard network protocols. These protocols include Telnet, HTML, IRC, and

raw TCP/IP socket level communication. A user of the chat server can take advantage of one or more of these connection objects to connect and communicate to the chat server and the other user objects.

Room objects gives the chat server the capability to divide the server into different communication areas. The room objects contain the different user objects as different users login to the chat server. Each room object has attributes that are presented to the user objects contained within the room object. These attributes include a URL frame, a rotating URL frame, a short description, a long description, and a real-time chat text area.

When a user logs on to the chat server, the user is assigned a user object and the preferences of that user are restored from a database. The user object is then moved into a room object which is either the last room that user visited or the default start room for that chat server. The user is then presented with the attributes of the room object along with the names of the other user objects within that room object. All of the user objects within the room objects see the same attributes and are given the capability to communicate with the other user objects within the room object.

The chat server is configurable to give the user ways to navigate between the different rooms on the server. Each room object can be configured with exits to other room objects. In this way, the chat server creates a topology of rooms that the user can navigate. The chat server can also be configured in a flat topology such that all room objects are immediately accessible to the user. All of the objects contains within the chat server have actions that are available to the user object to facilitate communication with other user objects. These actions include different methods of text communication to one or more users, viewing URLs between one or more users, and transferring files between users.

Spatial definition within the object oriented implementation calls for movement commands to navigate within the defined environment. A suitable set includes four categories

of movement commands based on function: directional movement, teleportation, following, and mobile privacy.

Directional movement is based on utilizing room exits to physically move one's avatar through the defined virtual space. The *go* command takes as an argument the direction in which the invoking user wishes to travel. Room exits become *go* commands when defined within the virtual space, in which case the user need only type the exit's name instead of preceding the name with the *go* command. Exit listings show up as hyperlinks which Web users need only point to and click to navigate, allowing even greater ease of movement.

The *teleportation* command is based on a user's jumping from one room to another in a fashion that omits the need to traverse intervening virtual space. The user's avatar simply leaves one room and enters another, whether or not connected. While teleportation is faster than directional movement, directional movement offers greater opportunities to meet new people and assist with more problems.

The *goto* command is a variation of the teleportation command. The *goto* command takes as an argument the name of a user to whom the invoker wishes to travel. Upon invocation, a request is sent to the target of the *goto* command to verify that it is acceptable for the teleporter to go to the target.

The *follow* command is based on users following other users from point-to-point within the virtual space. Since the followed user's avatar leaves one room and enters another by means of directional movement or teleportation, any following users move to the same destination using the same mode of travel. Followed Web users who encounter and utilize a hyperlink to an offsite destination are followed to that site by any followers who also have browser capabilities. Those without browser capabilities, *i.e.* telnet users, are simply left behind. Link following only works for the first offsite hyperlink encountered. The *follow* command takes one argument, which is the name of a user the invoker wishes to follow.

Upon invocation, a request is sent to the target of the *follow* command to verify that it is acceptable for the follower to follow the target.

Mobile privacy commands with regard to movement are based on one user granting or denying permission to others to follow or go to him or her. Five commands provide the means by which all users may insulate themselves from would-be hangers-on: autofollow, stopfollow, autogoto, allow, and disallow. The *autofollow* command is directly tied to the *follow* command. The *autofollow* command allows a user to set an automatic response to requests by other users to follow them, and takes a single argument selected from the following: ask, yes, and no. When the argument is "ask," any attempts to follow that user result in a notice being sent to that user informing him or her that another requests to be allowed to follow. At that time, the target of the follow, who received the message, responds with "allow" or "disallow" as desired. When *autofollow* is set to "yes" for a particular user, all requests to follow are automatically allowed. When *autofollow* is set to "no" for a particular user, all requests to follow are automatically disallowed. The *stopfollow* command is directly tied to the *follow* command. The *stopfollow* command allows a user who is following to stop following, and allows a user who is being followed to stop another form following him or her. The *autogoto* command is directly tied to the *goto* command, and allows a user to set an automatic response to requests by others to go to him or her. The *autogoto* command takes a single argument selected from the following: ask, yes, and no. When the argument is "ask," any attempts to go to that user result in a notice being sent to that user informing him or her that another requests to be allowed to go to his or her current location. At that time, the target of the go to, who received the message, responds with "allow" or "disallow" as desired. When *autogoto* is set to "yes" for a particular user, all requests to follow are automatically allowed. When *autogoto* is set to "no" for a particular user, all requests to follow are automatically disallowed. The *allow* command enables users to respond to incoming *goto* or *follow* requests, and takes as its single argument the name of the user to whom permission to use *goto* or *follow* should be granted. The *disallow* command

enables users to respond to incoming *goto* or *follow* requests, and takes as its single argument the name of the user to whom permission to use *goto* or *follow* should be denied.

Two additional movement commands are *move* and *suggest*. The *move* command, which is generally reserved for an administrator but which can be authorized to other users, allows this person to move one or more users into other rooms in the world. The *suggest* command is movement-like in that it allows a user a "peak" into another room or even another world by viewing an HTML page posted in the user's display window by the invoker of the command. The user and invoker remain in the current chat. The *suggest* command is also useful for displaying ad banners, which is achieved by designating the ad banner window default name *ichatad*.

The activity of managing and designing rooms in a virtual world and the various commands available are described in *i*chat ROOMS™ Administrator's Guide, IPN 960925.002.10.04, 1996, available from ichat, Inc. of Austin, Texas, which is hereby incorporated herein by reference in its entirety, and is included herein as an Appendix.

In an implementation of a chat server suitable for the Web, rooms are represented principally by separate URLs and are individual web-based (HTML) pages which may or may not be chat-enabled. Design of the URL to be displayed with each room is dictated primarily by the HTML designer's needs and the capabilities of the Web browser. Preferably, the chat and Web servers support all popular extensions, such as images, frames, plug-ins, Java® and JavaScript®, and ActiveX, and all popular multimedia extensions such as Real Audio, Shockwave, and Java Applets, and all can be synchronized with the chat and/or browse functions using the techniques described herein. Illustratively, the browser connection is an HTTP connection, and the chat connection is a telnet or IRC connection. Typically, an HTTP connection closes after document transfer completes, since current generation browsers such as Netscape® Navigator® Version 3.0 and Microsoft® Explorer® Version 3.0 require that it be so. However, alternative technologies such as server push require that the http connection be

maintained open, and may be used if desired. Typically, telnet and IRC connections remain open.

In an implementation of a chat server suitable for the Web, the chat client preferably is linked to the browser client for various reasons, including to establish a particular HTTP connection when the client is not the originator of the new connection request. For example, the chat server may need to cause a new HTTP connection to be made because a user object has just moved into a new room characterized by that connection. If the HTTP connection to the client is terminated, as is likely to be the case, the chat server cannot cause the HTTP server on the site to push the new document onto the browser client. However, the chat server is able to use the telnet or IRC connection to the chat client to make the request to load the new HTML page, and the request is then communicated by the chat client to the linked browser client which then pulls the new HTML page. If the HTTP connection happens to be open and the next HTML page is on the same site as the current HTML page, the browser server is able to push the HTML document onto the client browser if desired, or the transfer may rely on client pull.

Figure 6 and Figure 7 show an implementation of a world and the relationships between a chat server 610, an HTTP server 620, and the user's display, represented by a banner 630, an HTML page 640, and a chat 650. The world model of Figure 6 and Figure 7 is a simple two room world, for clarity, but is suitable for realizing the various events of Figure 4 occurring on the Fixit! site, as follows.

Arrival at the Fixit! site is shown in Figure 4C. At this time, the user object 616 is created in room object 614, which typically is an entry room. A user object 618 for Ralph the repair advisor resides in room object 612, but presently has nothing to do with the Ursula user object 616. Connections to the user object 618 are omitted for clarity. The user object 616 becomes fully aware of the web and chat connections that characterize the room object

614, thereby synchronizing the chat and browse functions. Other user objects associated with room object 614 are omitted for clarity.

The transition from Figure 4C to Figure 4D is caused by user Ursula clicking on the hyperlink "Chat with Rob the Receptionist" (action 421, Figure 4C). This activity occurs within room object 614.

The transition from Figure 4D to Figure 4E is executed by Rob the Receptionists, who issues a *move* command to move Ursula into room object 612 (Figure 7), which already has user object 612 resident. The user object 616 becomes fully aware of the web and chat connections that characterize the room object 612, thereby synchronizing the chat and browse functions. As a result, Ursula sees the HTML page 440 of Ralph the repair advisor and joins in the synchronized chat session 442.

The transitions from Figure 4E to Figure 4F, from Figure 4F to Figure 4G, and from Figure 4G to Figure 4H are executed by repair advisor Ralph using the *suggest* command to display new URLs 450, 460 and 470 on Ursula's browser screen while remaining in the object room 612 and continuing the same chat session 442. Each new URL is sent to the chat server 610, which sends information over the telnet or IRC connection to the chat application maintaining chat window 650, which in turn instructs the Web browser maintaining page 640 to load a new page. The chat application preferably uses local communications to the browser maintaining page 640 to load a new page corresponding to the new URL, although the chat application may be internal to the browser application if desired. A suitable local communications protocol is DDE, although other local communications protocols are suitable as well. The browser application establishes an HTTP connection to the appropriate server, which for this example is the HTTP server 620 although it could be a server on another site. Note that user object 616 remains in room object 612 (not shown).

The transition from Figure 4H to Figure 4I is executed by repair advisor Ralph using the *suggest* command to display new URLs 480 on Ursula's browser screen while remaining in the object room 612 and continuing the same chat session 442. This transition is similar to the transitions from Figure 4E to Figure 4F, from Figure 4F to Figure 4G, and from Figure 4F to Figure 4G, except that the HTTP connection is to a server on the BCD Computer site rather than to a server on the Fixit! site.

The transition from Figure 4I to Figure 4J is executed by repair advisor Ralph using the *follow* command to link his user object 618 to Ursula's user object 616, and then clicking on a hyperlink on the page 480 (action 481, Figure 4I) to join in a chat with the BCD Computer technician Terri. Following another user is achieved by locking together user objects, as shown by the tie bar 700 linking user object 616 to user object 618. Linked user objects 616 and 618 are extinguished from room object 612 but are resurrected in another room in the world on the BCD Computer site (not shown).

Advertising banners such as 446 (Figure 4E), 456 (Figure 4F), 466 (Figure 4G), 476 (Figure 4H), 486 (Figure 4I) and 630 (Figure 6 and Figure 7) are just small HTML windows and are handled in a similar fashion with one major exception. Preferably, banners are synchronized on a timed rotation, customizable by the second. The result is that users view multiple ad banners as often and for as long as configured by the Web site administrator.

Several well-known technologies are useful for embedding real-time chat into Web pages, either with or without synchronization of the chat and browser functions. The Netscape Navigator™ Web browser, available from Netscape Communications Corporation of Mountain View, California, has since the Beta release of version 2.0 in 1995 included a plug-in architecture. This plug-in architecture is a standard application programming interface that allows third party developers to write software programs, or modules, to extend the capabilities of the Web browser to view data types for which it might not have been designed to view. They are downloadable and become activate when needed. Other software

architectures accomplish similar results in somewhat different ways. For example, the Microsoft Explorer™ Web browser version 3.0, released in 1996, supports ActiveX controls. ActiveX controls are similar to plug-ins in what they can do and how they appear, but are different in that they download and install into the Web browser automatically.

An implementation of embedded chat suitable for use with the Netscape Navigator browser is shown in Figure 8. A user desiring to obtain the chat client software by download via the Web goes to a Web site that offers the software and clicks on a download button. This action causes a download request to be sent to an FTP server 810 (path 812). In response, the FTP server 810 (an HTTP server may be used instead, if desired) sends an executable file containing a self-extracting dynamic linking library file, or DLL file, to the user's computer (path 814). The user installs the chat plug-in DLL file in a plug-in subdirectory of the browser directory by executing the downloaded executable file. Installation of the chat plug-in is completed by having the user restart the client browser application, which registers the chat plug-in for future use as needed.

A user desiring to chat logs into a chat enabled site by completing a login page 820, which is furnished by an HTTP server 830 (path 832). Upon completion, the login page 820 identifies the user's name and password and contains an indication of the type of chat client software to be used, in this case a plug-in. It will be appreciated that some browsers such as version 3.0 of the Netscape Navigator browser are capable of generating the correct selection or limiting the number of displayed selections automatically. The data from the data fields of the completed form and the request for download is sent to the HTTP server 830 (path 834) when the "submit" button on the login page 820 is clicked. The HTTP server 830 passes the login data to a chat server 840 for an authorization check.

Once user chat authorization is granted, the HTTP server at that site, for example the HTTP server 830, sends an HTML page (path 836) that includes a sequence of HTML tags that causes a two frame window 850 to display on the user's monitor. The window 850 includes an HTML frame 852 and a plug-in frame 854 (see also Figure 4B). HTML pages

860 received by the browser client are recognized by the browser, which displays them in the HTML frame 852 in normal fashion. Chat files, which are assigned the MIME type application/x-chat or are identified by the extension .CHAT, are not processed by the browser since it does not natively support them. Instead, the browser consults its registry and invokes the chat plug-in to handle the .CHAT file type, so that chat pages 870 are displayed in the plug-in window 854. The HTML frame 852 and the plug-in frame 854 may or may not be synchronized, as desired.

A suitable sequence of HTML tags for creating the HTML frame 852 and the plug-in frame 854 in a Netscape Navigator browser is as follows.

```
<HTML>
...
<BODY>
<FRAMESET ROWS=*, 200>
<FRAME NAME=DISPLAY SRC=XXX.HTML>
<FRAME NAME=PLUGIN SRC=XXX.CHAT>
</FRAMESET>
...
</BODY>
</HTML>
```

The FRAMESET tag creates HTML and PLUGIN frames of a particular size, and the two FRAME tags identify the respective sources for the two frames.

Other approaches for embedding real time chat into a Web page using the Netscape Navigator browser include the EMBED command and helper applications, which are *per se* known in the art.

ActiveX controls are also useful for embedding real time chat into a Web page in a manner similar to plug-ins, although some of the details differ. ActiveX controls download and install into the Web browser automatically, and use a full screen "embed" like command to place HTML chat pages in the chat frame instead of .CHAT files. The manner in which the

capabilities of plug-ins can be replicated using ActiveX controls is generally well known in the art.

Java applets are also useful for embedding real time chat into a Web page. Java applets are delivered along with the HTML chat pages, and interpreted at runtime by an interpreter running on the user's computer.

An illustrative server architecture 900 suitable for handling a chat session using plug-ins and ActiveX controls, as well as Java applets in any combination is shown in Figure 9. Column 910 shows various clients, column 920 shows Web server 922 and Chat server 924, and column 930 shows various multimedia types. The architecture 900 is extendible to Netscape Navigator chat plug-ins 912 for Windows 95 and Macintosh System 7, ActiveX controls 914 for Microsoft Internet Explorer 3.0, Java clients (Unix and Others 916), and stand-alone clients for Microsoft Windows 3.1 (Unix and Others 916). This architecture provides seemless and platform independent communication using various multimedia types such as Real Audio 932, Shockwave 934, Java Applets 936, and Others 938 among people on chat-enabled web pages.

The description of the invention set forth herein is illustrative, and does not limit the scope of the invention as set forth in the following claims. Variations and modifications of the embodiments disclosed herein are possible. These and other variations and modifications of the embodiments disclosed herein may be made without departing from the spirit of the invention and from the scope of the invention as set forth in the following claims.